Practical Guidelines for Change Recommendation using Association Rule Mining

Leon Moonen* leon.moonen@computer.org
Stefano Di Alesio* stefano@simula.no
Dave Binkley‡ binkley@cs.loyola.edu
Thomas Rolfsnes* thomgrol@simula.no

* Simula Research Laboratory, Oslo, Norway
‡ Loyola University Maryland, Baltimore, Maryland, USA

ABSTRACT

Association rule mining is an unsupervised learning technique that infers relationships among items in a data set. This technique has been successfully used to analyze a system's change history and uncover *evolutionary coupling* between system artifacts. Evolutionary coupling can, in turn, be used to *recommend* artifacts that are potentially *impacted by a given set of changes* to the system. In general, the quality of such recommendations is affected by (1) the values selected for various parameters of the mining algorithm, (2) the characteristics of the change set used to derive the recommendation, and (3) the characteristics of the system's change history for which recommendations are generated.

In this paper, we empirically investigate the extent to which these factors affect change impact recommendations. Specifically, we conduct a series of systematic experiments on the change histories of two large industrial systems and eight large open source systems, in which we control the change size for which to derive recommendations, the measure used to assess the strength of the evolutionary coupling, and the maximum size of historical changes taken into account when inferring these couplings.We use the results from our study to derive a number of practical guidelines for applying association rule mining to derive software change impact recommendations.

Keywords

Evolutionary coupling, association rule mining, parameter tuning, change recommendations, change impact analysis.

1. INTRODUCTION

A well-know effect of the continued evolution of a software system is the increasing disorder or entropy in the system: as a result of repeated changes, the number and complexity of dependencies between parts of the code grows, making it increasingly difficult for developers to foresee and reason about the effects of changes they make to a system.

Automated *change impact analysis* techniques [7, 14, 26, 36] are aimed at supporting the developer during evolution by identifying the artifacts (e.g., files, methods, classes) that are affected by a given change. Traditionally, these change-impact analysis techniques are based on static or dynamic dependency analysis [6] (for example, by identifying the methods that call a changed method). More recently, promising alternative techniques have been proposed that identify dependencies by means of *evolutionary coupling*. These alternative approaches avoid certain limitations in existing techniques. For example, static and dynamic dependency analysis are generally language-specific, making them unsuitable for the analysis of heterogeneous software systems [34]. In addition, they can involve considerable overhead (e.g., dynamic analysis' need for code-instrumentation), and tend to over-approximate the impact of a change [25].

Evolutionary couplings differ from the ones found through static and dynamic dependency analysis, in that they are based on *how* the software was changed over time. In essence, evolutionary coupling aims to build on the developer's inherent knowledge of the dependencies in the system, which can manifest themselves by means of commit-comments, bug-reports, context-switches in an IDE, etc. In this paper, we consider *co-change* as the basis for uncovering evolutionary coupling. Co-change information can, for example, be extracted from a project's version control system [8], from its issue tracking system, or by instrumenting the development environment [27].

The most frequently used method for mining evolutionary coupling from co-change data is *association rule mining* (also called *association rule learning*) [1]. Various variants on the approach have been described in software engineering literature [38, 35, 17, 28]. All these approaches have in common that the technique is tuned with a number of parameters. While studying the literature, we found that there is little to no practical guidance on the tuning of these parameters, nor has there been a systematic evaluation of their effects on change-impact recommendation quality. Moreover, we conjecture that, in addition to parameters of the mining algorithm, the recommendation quality is also affected by characteristics of the change set that is used to derive the recommendation, and by characteristics of the system's change history from which recommendations are generated.

In this paper, we empirically investigate the extent to which these factors affect change impact recommendations. Specifically, we conduct a series of systematic experiments on the change histories of two large industrial systems and eight large open source systems. In these experiments we control the size of changes for which to derive recommendations, the measure used to assess the strength of evolu-

tionary coupling, and the maximum number of historical changes taken into account when inferring association rules. We use the results from our study to derive a number of practical guidelines for applying association rule mining to constructing software change impact recommendations.

Contributions: This paper presents three key contributions: (1) We investigate a previously unexplored area of tuning association rule mining parameters for software change impact recommendation. (2) We evaluate the impact of characteristics of the change set used to derive recommendations, and characteristics of change history used for learning, on recommendation quality. (3) We derive a number of practical guidelines for applying association rule mining to derive software change impact recommendations.

Overview: The remainder of this paper is organized as follows: Section 2 provides background on targeted association rule mining. Section 3 describes limitations of classical approaches. Section 4 describes the setup of our empirical investigation whose results are presented in Section 5. Finally, Section 6 presents the related work, and then Section 7 provides some concluding remarks.

2. ASSOCIATION RULE MINING

Agrawal et al. introduced the concept of *association rule mining* as the discipline aimed at inferring relations between *entities* of a dataset [1]. *Association rules* are implications of the form $A \rightarrow B$, where A is referred to as the *antecedent*, B as the *consequent*, and A and B are disjoint sets. For example, consider the classic application of analyzing shopping cart data: if multiple transactions include bread and butter then a potential association rule is *bread* $\rightarrow$ *butter*. This rule can be read as *"if you buy bread, then you are also likely to buy butter."*

In the context of mining evolutionary coupling from co-change information, the entities are the files of the system[1] and the collection (history) $\mathcal{T}$ of transactions, is the set of past *commits*. More specifically, a transaction $T \in \mathcal{T}$ is the set of files that were either changed or added while addressing a given bug or feature addition, hence creating a *logical dependence* between the files [9].

As originally defined [1], association rule mining generates rules that express patterns in a complete data set. However, some applications can exploit a more focused set of rules. *Targeted association rule mining* [31] focuses the generation of rules by applying a constraint. One example constraint specifies that the antecedent of all mined rules belongs to a particular set of files, which effectively reduces the number of rules that need to be created. This reduction drastically improves the execution time of rules generation [31].

When performing change impact analysis, rule-constraints are based on a *change set*, e.g., the set of modified files since the last commit, which is also known as the *query* for which to investigate the potential impact. In this case, only rules with at least one changed entity in the antecedent are created. The output of change impact analysis are the files from the system that are historically changed alongside the elements of the change set. For example, given the change set $\{a, b, c\}$, change impact analysis would uncover files that were changed when a, b, and c were changed. The resulting impacted files are those found in the rule consequents. These files can be ranked based on the rule's interestingness measure.

To our knowledge, only a few targeted association rule mining algorithms have been considered in the context of change impact analysis: Zimmerman et al. [38], Ying et al. [35], Kagdi et al. [17], and Rolfsnes et al. [28]. In contrast, simple *co-change* algorithms have been well studied in a variety of contexts [4, 5, 9, 12]. The existing targeted association rule mining algorithms and the simple co-change algorithms differ in terms of which subsets of the change-set are allowed in the antecedent of generated rules. Consider, for example, the subsets of the change-set $C = \{a, b, c, d\}$:

$$powerset(C) = \{\{\}, \quad (1)$$
$$\{a\}, \{b\}, \{c\}, \{d\}, \quad (2)$$
$$\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}, \quad (3)$$
$$\{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}, \quad (4)$$
$$\{a, b, c, d\}\} \quad (5)$$

Of C's subsets, both Zimmerman's and Ying's algorithms only consider rules based on line 5 (i.e., rules of the form $\{a, b, c, d\} \rightarrow X$) because these techniques constrain the antecedent to be equal to the change set. At the other end of the spectrum, co-change algorithms consider rules from the singleton sets in line 2, such as $\{a\} \rightarrow X$ or $\{b\} \rightarrow X$ (as a notational convenience we often write such rules without the set designation (e.g., $a \rightarrow X$). Rolfsnes et al. introduce TARMAQ, the most versatile among these existing algorithms. TARMAQ can use the sets from any of lines 2, 3, 4, or 5. The particular line used is dynamically chosen based on the maximal overlap with the change set [28].

3. PROBLEM DESCRIPTION

Change impact analysis takes as input a set of changed entities (typically files) in a system, referred to as a *change set* or *transaction*, and outputs a set of potentially impacted entities. A common strategy for change impact analysis is to use association rule mining to capture the *evolutionary couplings* between such entities. In particular, entities are considered coupled if they have changed together in the past, where the *strength* of a coupling depends on how frequently the entities changed together. The stronger the coupling between two entities, the more likely it is that the entities are related to each other, and hence that one is impacted by changes to the other.

The strength of evolutionary couplings is usually assessed using *interestingness measures* applied to the association rules. The literature reports over 40 of these measures, which have been applied in several domains [10]. However, it has been shown that the particular measure chosen has a significant impact over the ranking of rules, and consequently on the quality of recommendations generated [18, 32, 24].

Example 1 *Consider the following history $\mathcal{T}$ of transactions:*

$$\mathcal{T} = [\{a, c\}, \{b, c\}, \{a, b, c\}, \{a, b, x, y, z\}, \{y, z\}, \{x, y, z\}]$$

and the change set $C = \{a, b\}$ where, based on $\mathcal{T}$ and C, the following four rules have been mined. Three example measures,

[1]Observe that other levels of granularity are possible, and our consideration of co-change at the file level is without loss of generality, as these algorithms are granularity agnostic: provided that suitably co-change data is available (or computable), the algorithms will relate methods or variables just as well as files.

namely the confidence κ, prevalence ϕ, *and* recall ρ *of each rule are given in parentheses [18]. The confidence (recall) measures the ratio between the number of transactions where the antecedent and consequent changed, and the number of transactions where only the antecedent (consequent) changed. The prevalence measures the percentage of transactions in the history where the consequent changed.*

$$
\begin{aligned}
a, b \rightarrow c \quad & (\kappa = 1/2,\ \phi = 3/6,\ \rho = 1/3) \\
a, b \rightarrow x \quad & (\kappa = 1/2,\ \phi = 2/6,\ \rho = 1/2) \\
a, b \rightarrow y \quad & (\kappa = 1/2,\ \phi = 3/6,\ \rho = 1/3) \\
a, b \rightarrow z \quad & (\kappa = 1/2,\ \phi = 3/6,\ \rho = 1/3)
\end{aligned}
$$

The confidence values suggest that the four rules are all equally likely to hold, while the prevalence suggests that $a, b \rightarrow x$ is less likely to hold than the others, because x is changed only twice in $\mathcal{T}$. On the other hand, recall suggests the opposite, i.e., that $a, b \rightarrow x$ is more likely to hold than the others, because the co-occurrence of $\{a, b\}$ and x in $\{a, b, x, y, z\}$ is more significant than the co-occurrence of $\{a, b\}$ and other files that changed more often without both a and b. However, intuition matches the suggestion made by ϕ, i.e., that a and b have a stronger relation with c rather than with x. This is because c appears also singularly with a and b, while x tends to change together with y and z.

In general, the particular interestingness measure (or combination thereof) used to rank rules is a parameter of the change recommendation systems. There exist a number of such parameters that can affect the way rules are ranked and recommendations are generated and presented to the user.

Example 2 *Consider again the history $\mathcal{T}$ of transactions:*

$$\mathcal{T} = [\{a, c\}, \{b, c\}, \{a, b, c\}, \{a, b, x, y, z\}, \{y, z\}, \{x, y, z\}]$$

and the change set $C = \{a, b\}$. However, assume that transactions larger than 3 files are discarded from the history when generating rules and calculating their interestingness. Based on C and the filtered $\mathcal{T}$, only the rule $a, b \rightarrow c$ is mined.

In this second example, x, y, and z will not be recommended to the user, because $\{a, b, x, y, z\}$, the change set containing evidence for their recommendation, has been filtered out from $\mathcal{T}$. Intuitively, filtering large transactions from the history avoids generating rules from change sets that contain potentially unrelated files, and is a strategy used by most approaches [38, 35, 28]. However, filtering too aggressively could remove from the history legitimate evidence for the evolutionary coupling of files.

Values for configurable parameters are not the only criteria that can affect the recommendations generated.

Example 3 *Consider the same history $\mathcal{T}$ of transactions:*

$$\mathcal{T} = [\{a, c\}, \{b, c\}, \{a, b, c\}, \{a, b, x, y, z\}, \{y, z\}, \{x, y, z\}]$$

and the change sets $C_1 = \{a\}$ and $C_2 = \{a, b\}$. The following rules are mined for C_1 and C_2.

$$
\begin{aligned}
& a \rightarrow b \\
& a \rightarrow c \\
& a \rightarrow x \\
& a \rightarrow y \\
& a \rightarrow z
\end{aligned}
\qquad\qquad
\begin{aligned}
& a, b \rightarrow c \\
& a, b \rightarrow x \\
& a, b \rightarrow y \\
& a, b \rightarrow z
\end{aligned}
$$

C_1 and C_2 have different size, and hence, contain a different amount of evidence that suggests recommendations. In particular, C_1 contains only a which suggests that c is potentially impacted by the change. On the other hand, C_2 contains both a and b suggesting c. Intuitively, the size of the *query* also affects the recommendations generated, and in particular, the larger the query, the stronger the recommendation. However, queries which are too large may contain potentially unrelated files, for which generating good recommendations based on evolutionary coupling is hard.

A similar argument holds for the *expected outcome* of the query, i.e., the set of files that are actually impacted a change to the files of the query. Consider again Example 3, supposing that $\{a, b, c\}$ is the set of entities impacted by the changes in both C_1 and C_2. C_1 only contains a, for which both b and c have to be recommended. C_2 instead contains a and b, and only c has to be recommended. Intuition suggest that the smaller the expected outcome, the easier it is to generate recommendations. However, for particularly large expected outcomes it might be relatively easy to predict some of the potentially affected entities.

Finally, characteristics of the change history can also affect the precision of recommendations.

Example 4 *Consider two transaction histories $\mathcal{T}_\infty$ and $\mathcal{T}_\in$ of the systems $\mathcal{S}_\infty$ and $\mathcal{S}_\in$, respectively. Transactions in $\mathcal{T}_\infty$ are on average significantly larger than transactions in $\mathcal{T}_\in$ because of differences in the software processes used with $\mathcal{S}_\infty$ and $\mathcal{S}_\in$. For example, $\mathcal{S}_\infty$ might be developed with an agile process, where small change sets are committed frequently. In contrast, $\mathcal{S}_\in$ might be developed with a less nimble process, where large change sets are committed at a slower pace.*

Queries and expected outcomes are likely to reflect the average transaction size of the history they refer to. In particular, queries and expected outcomes from $\mathcal{T}_\infty$ are likely to be significantly smaller than those from $\mathcal{T}_\in$, entailing all the implications discussed in Example 3, and also potentially affecting the impact of transaction filtering discussed in Example 2.

4. EMPIRICAL STUDY

We perform a large empirical study to assess the extent to which the quality of change recommendations generated using targeted association rule mining is affected by (1) values selected for various parameters of the mining algorithm, (2) characteristics of the change set used to derive the recommendation, and (3) characteristics of the system's change history for which recommendations are generated. We use as a reference mining approach the work of Rolfsnes et al., which has proven to perform consistently better than the state-of the art alternatives for software change impact analysis [28]. Our study investigates the performance of targeted association rule mining in the context of several software-systems, and several parameters configurations.

Specifically, we investigate the following four research questions:

RQ 1 *To what extent does the interestingness measure affect the precision of change recommendation?*

RQ 2 *To what extent does the size limit used to filter the transactions of the history affect the precision of change recommendation?*

RQ 3 *To what extent do query size and expected outcome size affect the precision of change recommendation?*

RQ 4 *To what extent does the average commit (transaction) size in the history affect the precision of change recommendation?*

The remainder of this section details our evaluation setup, and is organized as follows: In Section 4.1 we explain the software-systems included in the study. Sections 4.2 and 4.3 describe the interestingness measures and the history filtering. In Section 4.4 we describe two central concepts for the evaluation, namely *query generation* and *query execution*. In Section 4.5 we explain the generation of change recommendations. In Section 4.6 we explain our method for measuring performance, and in Section 4.7 we introduce the implementation and execution environment of the evaluation.

4.1 Subject Systems

To assess targeted association rule mining in a variety of conditions, we selected ten large systems having varying size and frequency of transactions. Two of these systems come from our industry partners, namely Cisco Norway and Kongsberg Maritime (KM). Cisco Norway is the Norwegian division of Cisco Systems, a worldwide leader in the production of networking equipment. In particular, we consider a software product line for professional video conferencing systems developed by Cisco Norway. KM is a leading company in the production of systems for positioning, surveying, navigation, and automation of merchant vessels and offshore installations. Specifically, we consider a common software platform KM uses across various systems in the maritime and energy domain. The other eight systems are from well known open-source projects, namely Git, Apache HTTP Server (HTTPD), JetBrains IntelliJ IDEA (JetBrains), the Linux Kernel, Low-Level Virtual Machine (LLVM), Ruby on Rails, Apache Subversion, and Wine.

Table 1 summarizes descriptive characteristics of the software systems used in the evaluation. The table shows that the systems we selected vary from medium to large size, with up to forty thousand different files committed in the transaction history. For each system, we considered the 30000 most recent transactions (*commits*). We argue that this value represents a balance between a too short history, which would lack sufficient connections, and a too long history, which is hard to process efficiently and can contain couplings that do not hold anymore because of architectural changes. Across all ten systems, 30000 transactions cover a significantly differ time span of the development history, ranging from almost 20 years in the case of HTTPD, to 6 months in the case of the Linux kernel. Most of the systems are heterogeneous, i.e., they are developed in more than a one programming language.

4.2 Interestingness Measures

Applications in software change impact analysis often use or combine *support* and *confidence* measures from the data mining community to differentiate interesting from uninteresting rules [38, 35, 16, 28]. However, as introduced in Section 3, over 40 interestingness measures have been defined in the literature to measure the strength of the evolutionary couplings mined through association rules. These measures are usually defined based on a probabilistic interpretation of the occurrence in the history of the rules antecedent and consequent. For example, given the rule $A \rightarrow B$ the probability $P(A)$ is the percentage of transactions in the history that contain A, while the probability $P(A, B)$ is the percentage of transactions in the history that contain both A and B. Therefore, the interestingness of the rule $A \rightarrow B$ is usually defined as a function of $P(A)$, $P(B)$, and various combinations thereof obtained through negations, fractions, and conditional operators. In this paper, we consider 39 interestingness measures commonly used in several data mining and machine learning applications. Due to space restrictions, we only report their name in Table 2, and refer the reader to the sources where they are defined [21, 29].

4.3 History Filtering

Several approaches to software change impact analysis using data mining filter the history to remove transactions larger than a given size. This is a common heuristic used to avoid mining from transactions that do not contain relevant information on the evolutionary coupling of files, such as in the case of license updates or refactoring [35, 38, 2, 16]. In this paper, we consider seven different transaction filtering sizes: 2, 4, 6, 8, 10, 20, and 30. For each of transaction filtering size s, we generate a filtered history H_s, from which we mine the association rules and calculate their interestingness score to generate recommendations. In addition, we also compare recommendations yield by these seven filtering thresholds with those generated from the unfiltered history H. Note that, in the rest of these paper, we simply refer to the seven thresholds and the case of the unfiltered history as the *eight* transaction filtering sizes.

4.4 Query Generation and Execution

Conceptually, a *query* Q represents a set of files that a developer changed since the last synchronization with the version control system. The key idea behind our evaluation is to generate, starting from a transaction T, a query that emulates a developer erroneously forgetting to update some subset of T. Thus we mimic a developer forgetting to change one or more files. To do so, we randomly partition each transaction T into a non-empty query Q and a non-empty expected outcome $E \stackrel{\text{def}}{=} T \setminus Q$. In this way, we can evaluate the capability of a recommendation to infer E from Q.

From each system history H, we sample 300 transactions, and for each of those we generate a single query. Note that, in order to generate non-empty queries, we only sample from commits larger than a single file. The resulting 300 queries are executed 312 times each, once for each combination of the 39 interestingness measures reported in Table 2 and the 8 filtering sizes described in Section 4.3. This setup yields a total of $300 \cdot 312 = 93600$ data points for each system, where each data point is a recommendation for a query.

4.5 Generating Change Recommendations

All queries are executed using TARMAQ, the targeted association rule mining algorithm introduced by Rolfsnes et al. [28]. Recall from Section 2 that executing a query Q creates a set of association rules. Generating a change recommendation for Q requires sorting the rules generated for Q according to their interestingness score, and returning the ranked list of the consequents of the rules. Note that we consider only the largest interestingness score for each con-

Table 1: Characteristics of the evaluated software systems

Software System	Unique files	Avg. transaction size (# files)	History covered by 30 000 transactions	Languages used*
Cisco Norway	41701	6.20	1.07 years	C++, C, C#, Python, Java, XML, other build/config (% undisclosed)
Kongsberg Maritime	35111	5.08	15.97 years	C++, C, XML, other build/config (% undisclosed)
Git	3574	1.95	10.42 years	C (45%), shell script (35%), Perl (9%), 14 other (11%)
HTTPD	10021	4.80	19.78 years	XML (56%), C (32%), Forth (8%), 19 other (4%)
Linux Kernel	19768	2.14	0.48 years	C (94%), 16 other (6%)
LLVM	20745	4.41	2.15 years	C++ (71%), Assembly (15%), C (10%), 16 other (4%)
JetBrains	36162	3.38	1.58 years	Java (71%), Python (17%), XML (5%), 26 other (7%)
Ruby on Rails	5346	2.25	5.78 years	Ruby (98%), 6 other (2%)
Subversion	2915	2.76	7.61 years	C (61%), Python (19%), C++ (7%), 15 other (13%)
Wine	6679	2.47	4.61 years	C (97%), 16 other (3%)

* data on the languages used by the open source systems obtained from http://www.openhub.net.

Table 2: Overview of the 39 interestingness measures considered in our study

#	Interestingness Measure
1	Added Value
2	Casual Confidence
3	Casual Support
4	Collective Strength
5	Confidence
6	Conviction
7	Cosine
8	Coverage
9	Descriptive Confirmed Confidence
10	Difference Of Confidence
11	Example and Counterexample Rate
12	Gini Index
13	Imbalance Ratio
14	Interestingness Weighting Dependency
15	J Measure
16	Jaccard
17	Kappa
18	Klosgen
19	Kulczynski
20	Laplace Corrected Confidence
21	Least Contradiction
22	Leverage
23	Lift
24	Linear Correlation Coefficient
25	Loevinger
26	Odd Multiplier
27	Odds Ratio
28	One Way Support
29	Prevalence
30	Recall
31	Relative Risk
32	Sebag Schoenauer
33	Specificity
34	Support
35	Two Way Support
36	Varying Rates Liaison
37	Yules Q
38	Yules Y
39	Zhang

sequent. This means that, for the purpose of this paper, we do not consider rule aggregation strategies, such as those proposed by Rolfsnes et al. [29].

4.6 Performance Measure

To evaluate a recommendation we use the *average precision* (AP) measure:

Definition 1 (Average Precision) *Given a recommendation* R*, and an expected outcome* E*, the average precision of* R *is given by:*

$$AP(R) \stackrel{def}{=} \sum_{k=1}^{|R|} P(k) * \triangle r(k) \quad (6)$$

where $P(k)$ is the precision calculated on the first k files in the list (i.e., the fraction of correct files in the top k files), and $\triangle r(k)$ is the change in recall *calculated only on the $k-1^{th}$ and k^{th} files (i.e., how many more correct files where predicted compared to the previous rank).*

Note that since we consider only rules with single consequents, $\triangle r(k)$ will always be equal to either zero or $1/|E||$, i.e., a rank either does not contain a file from the expected outcome, or it contains exactly one file from the expected outcome. Table 3 illustrates the computation of AP, $P(k)$, and $\triangle r(k)$ given the ranked list $[c, a, f, g, d]$ and the expected outcome $\{c, d, f\}$.

Table 3: Example of average precision calculation

Consider as relevant files: c, d, f

Rank (k)	File	$P(k)$	$\triangle r(k)$
1	c	1/1	1/3
2	a	1/2	0
3	f	2/3	1/3
4	g	2/4	0
5	d	3/5	1/3

$AP = 1/1 * 1/3 + 1/2 * 0 + 2/3 * 1/3 + 2/4 * 0 + 3/5 * 1/3 \approx 0.75$

As an overall performance measure of a group of factors, e.g., a given filtering size using a given interestingness measure, we use the *mean average precision* (MAP) computed over all the queries executed using that factor combination.

4.7 Experimental Setup

The rule generation algorithm and interestingness measures have been implemented in Ruby. We performed the experiment on Intel Xeon processors running at 2.50Ghz.

5. RESULTS

This section presents the results of the study described in Section 4. We first analyzed the key descriptive statistic for the

systems, i.e., the distribution of the commit size shown in Table 4. Because the majority (90+%) of the commits contains less than 6 files, we focus the remainder of our analysis on this dominant subset of the data.

First, we performed an ANOVA using the five explanatory variables system, filter size, expected outcome size, measure, and query size, together with all the ten possible pairwise interactions, i.e. a total of 15 terms. The Q-Q Plot of the residuals, omitted due to space restrictions, shows that the residuals follow a sufficiently normal distribution, especially considering ANOVA's performance in the presence of large data sets such as the nearly one million data points used in this study. The resulting model is shown in Table 5, where terms are ordered based on F-value. Even though all 15 terms from the model are highly statistically significant, the five variables have the largest contribution (the largest F-value).

From the resulting model we draw three major conclusions: 1) the various interestingness measures show consistent and independent behavior, 2) aggressive filtering of the history provides the best performance, and 3) the interrelationship between query size expected outcome size captures the expected pattern that favors large queries and small expected outcomes.

We being by considering RQ1: "To what extent does the interestingness measure affect the precision of change recommendation?" To consider the influence of interestingness measure, Figure 1 shows interaction plots of measure with system, query size, expected output size, and filter size. The overall pattern, evident in these graphs, is that measure is largely independent of system, query size, expected output size, and filter size. Statistically, the interactions are significant primarily due to a few interestingness measures that "buck the trend" (producing line crosses in the plots). This "mostly independent" observation is supported by the comparatively low F-values for the four interactions involving "measure" (the four are four of the five smallest in the model shown in Table 5).

The low F-values and the interactions plots support the notion that there are consistent best measures for software recommendations based on evolutionary coupling. Not surprisingly, there is no single best measure. Using Tukey's honestly significant difference test, 14 measures populate the top equivalence class. These are shown in Table 6.

Note the Table 6 includes the classic measures m_confidence and m_support. Their presents reinforces a result from the recent work of Le and Lo [21]. While their work considers a slightly different problem (the effect of different interestingness measures in rule-based specification mining), they too conclude that standard measures work well enough. Thus in summary for RQ1, we find that to a large extent measure's influence on precision is consistent across differences in other variables and that traditional measures are top performers.

Our second major conclusion relates to research questions RQ2 and RQ4. The second research question, RQ2 asks "To what extent does the size limit used to filter the transactions of the history affect the precision of change recommendation?" while RQ4 is "To what extent does the average transaction size in the history affect the precision of change recommendation?" The results for these questions are surprising. When filtering the history it is common to removes large commits as they tend to reflect licencing changes and alike. Prior work has typically used 30 as a cut off, while experiments have used a value as high as 100. For the 8 filter sizes studied, Figure 2 shows the interaction between filter size and system. It is clear from this figure that smaller commits contain much more useful information than larger commits as the best value for most systems occurs at a filter size of only 4. Statistically, Tukey's honestly significant difference test groups filter size of 4, 6, and 8 in the top equivalence class with means of 0.369, 0.368, and 0.365, respectively. Thus, in answer to RQ2, aggressive history filtering appears to retain only high value commits that support the creating of high quality association rules.

The most notable expectation, km has the largest average commit size. If the figure is redrawn scaling each system's data by its average commit size then km comes in line with the others. Thus in answer to RQ4 Average transaction size appears to be a normalizer when considering the impact of filter size on the precision of change recommendation?

This result is very interesting. It suggests that filtering should be much more aggressive. Despite smaller commits capturing more focused information, prior work has not been able to exploit the precision. For example ROSE [38] include a subset requirement that basically forces the algorithm to require large commits. In contrast, the more recent TARMAQ algorithm [28] can exploit partial information and thus can be used with only the high-precision commits.

Finally, because all the explanatory variables *and* all their interactions are statistically significant, the resulting coefficient equation is very complex (in includes almost 100 terms!). Rather than state this equation, we focus in on a few key terms. Specifically those involving query size, expected outcome size and commit size, which is the sum of query size and expected outcome size. These three include two significant interactions and thus their interpretation is complex. Fortunately the range of these explanatory variables is limited and thus it is possible to enumerate the possible combinations. Scaled to integers to show the relative contribution, Table 7 unravels the interactions. Based on the numbers, the easiest case (where the MAP value expects the greatest increase) is with a query size of 4 and an expected outcome size of 1, while the hardest case (where the MAP value expects the greatest decrease) is the opposite corner where query size is 1 and expected outcome size is 4. From the table it is clear that for a fixed query size (any column of the table) the prediction gets harder (the MAP value decreases) as expected outcome size grows. Look at by row, for a fixed expected outcome size, the prediction gets easier as the query size increases.

Again turning to Tukey's test, for query size the highest MAP value is obtained using the value 2, while for expected outcome size the highest MAP value is obtained at the value 1 (i.e., when predicting a single element).

5.1 Threats to validity

Commits as a basis for evolutionary coupling: The evaluation presented in this paper is grounded in predictions made from analyzing patterns in change-histories. The transactions that make up the change-histories are however not in any way guaranteed to be "correct" or "complete", in the sense that they represent a coherent unit of work. Non-related files may be present in the transactions, and related files may be missing from the transactions. However, the included software-systems in our evaluation all (except KM) use Git for version control. As Git provides developers with tools for amending commits and rewriting history,

Table 4: Cumulative Percent of data considered for each commit size

commit size	1	2	3	4	5	6	7	8	9	10	all
commits	157636	48903	23801	13767	8561	5546	3829	2814	2103	1611	280179
cumulative percent of data	56.3%	73.7%	82.2%	87.1%	90.2%	92.2%	93.5%	94.5%	95.3%	95.9%	100%

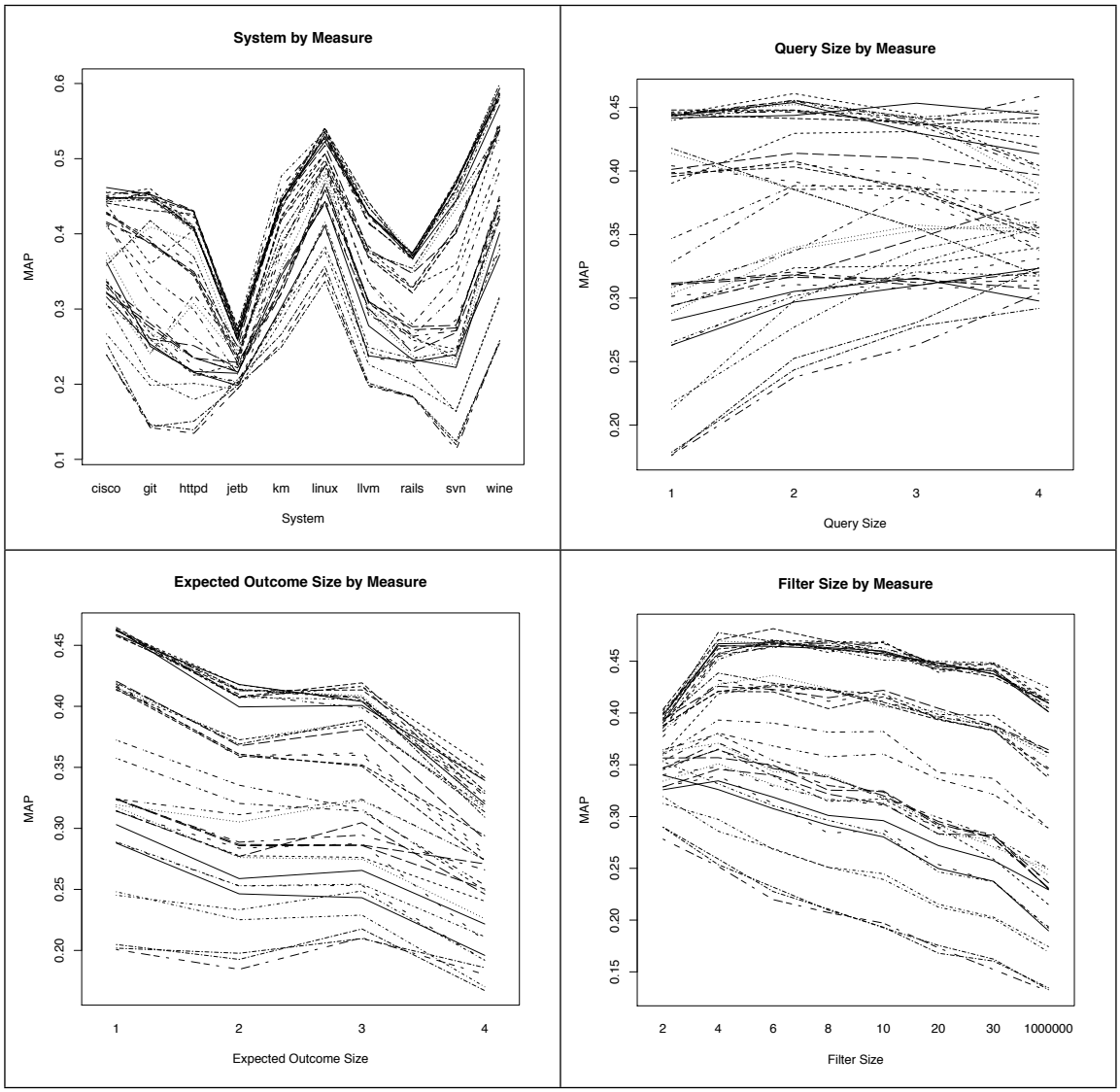

Figure 1: Interaction plots of interestingness measure, and system, query size and expected output size. The measures are shown unlabeled to avoid cluttering the plot.

Table 5: Statistical Model - Explanatory variables are placed atop interactions and then each ordered by F-value, which provides a measure of the significance of the given term.

Resulting ANOVA Model				
Explanatory Variable	Df	Mean Sq	F-value	p-value
program	9	332	2369.70	< 0.0001
filter size	1	289	2065.60	< 0.0001
expected outcome size	1	219	1566.45	< 0.0001
measure	39	95	675.35	< 0.0001
query size	1	26	188.83	< 0.0001
program:query size	9	25	175.79	< 0.0001
program:expected outcome size	9	20	143.95	< 0.0001
expected outcome size:filter size	1	16	115.45	< 0.0001
program:filter size	9	8	58.26	< 0.0001
query size:expected outcome size	1	7	47.17	< 0.0001
query size:measure	39	3	24.64	< 0.0001
query size:filter size	1	3	21.88	< 0.0001
program:measure	351	1	10.47	< 0.0001
expected outcome size:measure	39	1	5.87	< 0.0001
filter size:measure	39	1	6.19	< 0.0001

Table 6: Top Interestingness Measures

interestingness measure	MAP
m_confidence	0.447
m_casual_confidence	0.447
m_descriptive_confirmed_confidence	0.446
m_loevinger	0.446
m_example_and_counterexample_rate	0.446
m_leverage	0.445
m_klosgen	0.445
m_collective_strength	0.444
m_added_value	0.444
m_disjunct_confidence	0.443
m_support	0.443
m_difference_of_confidence	0.442

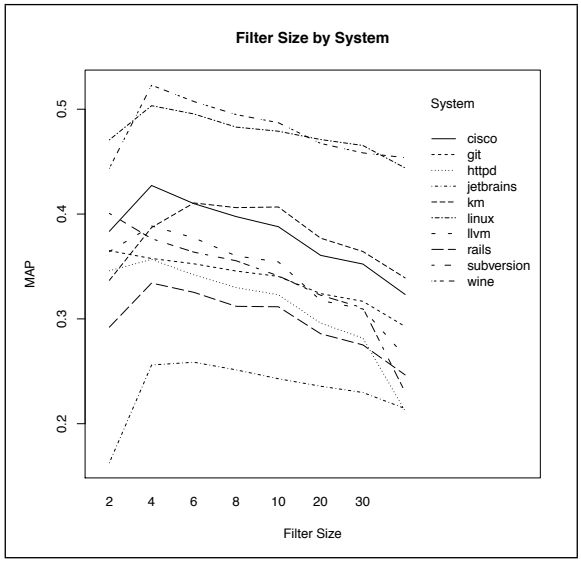

Figure 2: Interactions of filter size and system.

Table 7: Impact of query size and expected outcome size

expected outcome size	query size 1	2	3	4
1	-24	-8	12	32
2	-55	-32	11	
3	-74	-24		
4	-85			

we do believe that there are less issue than with other versioning systems such as SVN or CVS. In our related work we discuss methods for grouping related commits that are orthogonal to our approach and could be used to prepare a refined change-history.

Variation in software systems: We conducted our experiments on two industrial systems and eight large open source systems. These systems vary considerably in size and frequency of transactions (commits), which should provide an accurate picture of the performance in various contexts (see Table 1). However, despite our careful choice, we are likely not to have captured all possible variations.

Random sampling errors: Our experiment is based on taking a large number of simple random samples from the change history of each system. Although we use uniform random sampling, there is the possibility that our random samples do not accurately represent the actual change history, for example the distribution of transaction sizes considered in the sample may be different than the distribution of transaction sizes in the full history. We address this issue by conducting statistical tests (chi-squared test) to validate that our samples are representative of the change histories. An alternative approach would be to use a stratified sampling strategy where one takes an amount of samples of each transaction size that is proportional to the frequency of that transaction size in the complete history. The results of our statistical tests indicate that such an alternative strategy was not required.

Implementation: We have implemented the experiments

using Ruby and thoroughly tested all algorithms and measures studied in this paper. We also performed the statistical analysis using standard methods provided by R. However, we can not guarantee the absence of implementation errors which may have affected our results.

6. RELATED WORK

Recent research highlighted that the performance of data mining algorithms is affected by their configuration parameters [23]. A common strategy for tuning these parameters consists in optimizing their values over half of the test set (*inner* validation), and using the other half to assess the performance of the optimal parameters (*outer* validation) [33]. While this strategy is useful to fine-tune the parameters for a particular type of data, it does not provide insights on how the algorithm performance is affected by such parameters [19]. In the context of association rule mining, several influential authors state the problem of investigating the parameters sensitivity of the quality of generated rules [37, 22, 15].

Therefore, in this paper, we investigate the extent to which the quality of software recommendations derived via association rule mining is affected by three factors, namely (1) values selected for various parameters of the mining algorithm, (2) characteristics of the change set used to derive the recommendation, and (3) characteristics of the system's change history for which recommendations are generated. In the rest of this Section, we distinguish related work on these three aspects, focusing on the area of software maintenance and evolution.

Parameters in Association Rule Mining: In general, association rule mining algorithms differ from each other for the data structures used to represent transactions, and the strategy that selects transactions relevant for a given query [30]. However, the majority of such algorithms is characterized by similar parameters. Among those, this paper focuses on the maximum size of transactions used to generate rules (*transaction filtering size*), and on the metric measuring the strength of evolutionary couplings inferred by those rules (*interestingness measure*). In the context of software change impact analysis, several studies remark the importance of discarding from the history large change sets which are likely to contain unrelated files. For example, Kagdi et al [16], Zimmermann et al. [38] and Ying et al. [35] propose to filter out transactions larger than 10, 30, and 100 items, respectively. However, the authors do generally not report how such threshold values have been chosen, and do not argue on whether different values could affect the precision of recommendations. Several authors also remark how selecting the right interestingness measure for a problem domain can significantly affect recommendation accuracy [32, 21, 24, 10]. In particular, Le and Lo compared 38 measures in the context of rule-based specification mining, highlighting the need to look beyond standard support and confidence to find interesting rules [21]. We are not aware of similar studies carried out in the context of software change impact analysis.

Characteristics of the Change Set: Targeted association rule mining approaches drive the generation of rules by a *query* supplied by the user [31]. In general, particular characteristics of the query can effect the precision of recommendations. For example, in the context of software change impact analysis, Rolfsnes et al. identified a particular class of queries for which the most common targeted association rule mining approaches cannot generate recommendations [28]. Note that it is common practice to validate targeted association rule mining approaches by sampling random queries from the change history [38, 35]. While this strategy ensures that the approach is evaluated on a variety of change sets in the history, authors do not in general argue on whether the size of such queries significantly affects the quality of recommendations. Among others, Hassan and Holt investigated the effectiveness of evolutionary coupling in predicting change propagation effects resulting from source code changes, but did not evaluate whether the size of transactions in the history affects the quality of the predictions generated [12].

Characteristics of the Change History: Over the years, several studies proposed strategies to group transactions in the revision history of software projects [13, 38, 17]. The reason for doing so is that a developer might accidentally commit an incomplete transaction, and modify the remaining files related to the same change in a separate transaction. As a consequence, a single change set might be scattered across several transactions in the change history. Nevertheless, in modern version control transactions are stashed in the user local repository and finalized at a later stage, reducing the risk of committing incomplete transactions. In contrast, whether properties of the change history such as average commit size and frequency affect the quality of software recommendations is a relatively less studies subject. In this direction, German carried out an empirical study on several open source projects, finding that the revision history of most systems contains mostly small commits [11]. Alali et al. also investigated the total number of lines modified in the files, and the total number of hunks with line changes [3]. Kolassa et al. performed a similar study on commit frequency, reporting an average inter-commit time around three days [20]. However, none of these studies investigates how characteristics of the change history affect the quality of change recommendations.

7. CONCLUDING REMARKS

Association rule mining is an automated, unsupervised, learning technique that has been successfully used to analyze a system's change history, and uncover *evolutionary coupling* between its artifacts. These evolutionary couplings can be used to *recommend* artifacts that are potentially *impacted by a given set of changes* to the system, which helps developers address the increasing entropy caused by repeated software maintenance and evolution tasks.

In general, the quality of such recommendations is affected by (1) values selected for various parameters of the mining algorithm, such as filtering of transaction size, interestingness measure (2) characteristics of the change set used to derive the recommendation, such as query size w.r.t. expected outcome size, and (3) characteristics of the system's change history for which recommendations are generated, such as transaction size.

In this paper, we have empirically investigated the extent to which these factors affect change impact recommendations. Specifically, we have conducted a series of experiments

on the change histories of two large industrial systems and eight large open source systems. For each of these, we randomly took a representative sample of the transactions in the change history, randomly split each of these transactions in two parts, respectively a query and an expected outcome, and analyse how the parameters discussed above affect the prediction of the expected outcome based on the query. We use the results from our study to derive a number of practical guidelines for applying association rule mining to derive software change impact recommendations.

We draw the following conclusions: First, our analysis shows that the particular system, query size and expected outcome size do not significantly affect the quality of the recommendations made using the different interestingness measures. This means that it is possible to assess what are the best measures for software recommendations based on evolutionary coupling. To do so, we clustered the measures into into groups such that measures in each group are not statistically different from each other. Our conclusion is that standard measures such as confidence and support are in the top group, even though they are not the absolute best ones. This is in line with earlier findings by Lo and Le [21], who conducted an empirical study on interestingness measures for rule-based specification mining.

Second, learning from a change history that is filtered to contain a maximum transactions between 4 and 6 yields the best precision, regardless of system size, query size, and expected outcome size, and the difference in precision compared to these higher thresholds is significant. In general, the actual value seems to be dependend of the average commit size, as exemplified by the fact that one of the subject systems (km) with a significantly larger average commit size than the others also performs better with filtering on a larger maximum transaction size. Our future work includes a further investigation of the interaction between the average commit size and optimal filtering size.

Third, up to a transaction size of around five (which includes just over 90% of the transactions in our data), the larger the query, the higher the average precision of predicting the missing part of the transaction. This suggests that for relatively small commits, the more information you have, the better the results If we extend the analysis to larger commits (which are the minority), large queries lead to lower precision. These points suggests that queries from small commits are likely to contain files related to each other, and for which recommendations are precise. On the other hand, large queries from large commits are likely to contain files unrelated to each other, and for which recommendations are not (as) precise. This is in accordance with the results in our previous conclusion on the filtering size.

Four, up to a transaction size of around five (which includes just over 90% of the transactions in our data), the smaller the expected outcome, the higher the average precision of predicting that outcome. This suggests that for relatively small commits the less information you want to predict, the better the results If we extend the analysis to larger commits (which are the minority), larger expected outcomes lead to higher precision. These points suggests that predicting a low number of files is easy, while predicting a higher number of files from limited information is hard, which conforms to what can be intuitively expected.

We conclude with the following practical guidelines for applying association rule mining in the context of software change impact recommendations: (1) stick to default interestingness measures, although not always optimal, they perform among the best in class, (2) learn from a subset of the change history that includes only the smaller transaction sizes to avoid noise. We have good experience with transaction sizes up to 5 or 6, but these values may depend on the actual transaction sizes in the particular history that is analyzed.

Acknowledgement: This work is supported by the Research Council of Norway through the EvolveIT project (#221751/F20) and the Certus SFI. Dr. Binkley is supported by NSF grant IIA-1360707 and a J. William Fulbright award.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. "Mining association rules between sets of items in large databases". In: *ACM SIGMOD International Conference on Management of Data*. ACM, 1993, pp. 207–216.

[2] A. Alali. "An Empirical Characterization of Commits in Software Repositories". Ms.c. Kent State University, 2008, p. 53.

[3] A. Alali, H. Kagdi, and J. Maletic. "What's a Typical Commit? A Characterization of Open Source Software Repositories". In: *2008 16th IEEE International Conference on Program Comprehension*. IEEE, 2008, pp. 182–191.

[4] T. Ball, J. Kim, and H. P. Siy. "If your version control system could talk". In: *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*. 1997.

[5] D. Beyer and A. Noack. "Clustering Software Artifacts Based on Frequent Common Changes". In: *13th International Workshop on Program Comprehension (IWPC)*. IEEE, 2005, pp. 259–268.

[6] S. Bohner and R. Arnold. *Software Change Impact Analysis*. CA, USA: IEEE, 1996.

[7] G. Canfora and L. Cerulo. "Impact Analysis by Mining Software and Change Request Repositories". In: *11th IEEE International Software Metrics Symposium (METRICS)*. IEEE, 2005, pp. 29–37.

[8] S. Eick et al. "Does code decay? Assessing the evidence from change management data". In: *IEEE Transactions on Software Engineering* 27.1 (2001), pp. 1–12.

[9] H. Gall, K. Hajek, and M. Jazayeri. "Detection of logical coupling based on product release history". In: *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1998, pp. 190–198.

[10] L. Geng and H. J. Hamilton. "Interestingness measures for data mining". In: *ACM Computing Surveys* 38.3 (2006).

[11] D. M. German. "An empirical study of fine-grained software modifications". In: *Empirical Software Engineering* 11.3 (2006), pp. 369–393.

[12] A. Hassan and R. Holt. "Predicting change propagation in software systems". In: *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* IEEE, 2004, pp. 284–293.

[13] F. Jaafar et al. "Detecting asynchrony and dephase change patterns by mining software repositories". In: *Journal of Software: Evolution and Process* 26.1 (2014), pp. 77–106.

[14] M.-A. Jashki, R. Zafarani, and E. Bagheri. "Towards a more efficient static software change impact analysis method". In: *8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*. ACM, 2008, pp. 84–90.

[15] N. Jiang and L. Gruenwald. "Research issues in data stream association rule mining". In: *ACM SIGMOD Record* 35.1 (2006), pp. 14–19.

[16] H. Kagdi, M. Gethers, and D. Poshyvanyk. "Integrating conceptual and logical couplings for change impact analysis in software". In: *Empirical Software Engineering* 18.5 (2013), pp. 933–969.

[17] H. Kagdi, S. Yusuf, and J. I. Maletic. "Mining sequences of changed-files from version histories". In: *Proceedings of the 2006 international workshop on Mining software repositories - MSR '06*. New York, New York, USA: ACM Press, 2006, p. 47.

[18] R. Kamber, Micheline and Shinghal. "Evaluating the Interestingness of Characteristic Rules". In: *KDD*. 1996, pp. 263–266.

[19] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. "Towards parameter-free data mining". In: *Sigkdd* (2004), pp. 206–215.

[20] C. Kolassa, D. Riehle, and M. A. Salim. "The empirical commit frequency distribution of open source projects". In: *Proceedings of the 9th International Symposium on Open Collaboration - WikiSym '13*. New York, New York, USA: ACM Press, 2013, pp. 1–8.

[21] T.-d. B. Le and D. Lo. "Beyond support and confidence: Exploring interestingness measures for rule-based specification mining". In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 331–340.

[22] W. Lin, S. A. Alvarez, and C. Ruiz. "Efficient adaptive-support association rule mining for recommender systems". In: *Data Mining and Knowledge Discovery* 6 (2002), pp. 83–105.

[23] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. 2010, p. 1383.

[24] K. Mcgarry. "A survey of interestingness measures for knowledge discovery". In: *The Knowledge Engineering Review* 20.01 (2005), p. 39.

[25] A. Podgurski and L. Clarke. "A formal model of program dependences and its implications for software testing, debugging, and maintenance". In: *IEEE Transactions on Software Engineering* 16.9 (1990), pp. 965–979.

[26] X. Ren et al. "Chianti: a tool for change impact analysis of java programs". In: *ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 2004, pp. 432–448.

[27] R. Robbes, D. Pollet, and M. Lanza. "Logical Coupling Based on Fine-Grained Change Information". In: *Working Conference on Reverse Engineering (WCRE)*. IEEE, 2008, pp. 42–46.

[28] T. Rolfsnes et al. "Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis". In: *23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2016, p. 12.

[29] T. Rolfsnes et al. "Improving Change Recommendation using Aggregated Association Rules". In: *13th International Conference on Mining Software Repositories (MSR) (to appear)*. 2016.

[30] A. Silva and C. Antunes. "Constrained pattern mining in the new era". In: *Journal of Knowledge and Information Systems* online (2015), pp. 1–28.

[31] R. Srikant, Q. Vu, and R. Agrawal. "Mining Association Rules with Item Constraints". In: *International Conference on Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[32] P.-N. Tan, V. Kumar, and J. Srivastava. "Selecting the right objective measure for association analysis". In: *Information Systems* 29.4 (2004), pp. 293–313.

[33] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.

[34] A. R. Yazdanshenas and L. Moonen. "Crossing the boundaries while analyzing heterogeneous component-based software systems". In: *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 193–202.

[35] A. Ying et al. "Predicting source code changes by mining change history". In: *IEEE Transactions on Software Engineering* 30.9 (2004), pp. 574–586.

[36] M. B. Zanjani, G. Swartzendruber, and H. Kagdi. "Impact analysis of change requests on source code based on interaction and commit histories". In: *Working Conference on Mining Software Repositories (MSR)* (2014), pp. 162–171.

[37] Z. Zheng, R. Kohavi, and L. Mason. "Real world performance of association rule algorithms". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*. New York, New York, USA: ACM Press, 2001, pp. 401–406.

[38] T. Zimmermann et al. "Mining version histories to guide software changes". In: *IEEE Transactions on Software Engineering* 31.6 (2005), pp. 429–445.